Trabajo Final

Ingeniería del Software

INTEGRANTES: Candotti, Enzo
Flores, Mauricio
Nicolaide, Christian

Profesor: **Mgr. Martín Miceli**
Ing. Julian Nonino

Nota de Entrega

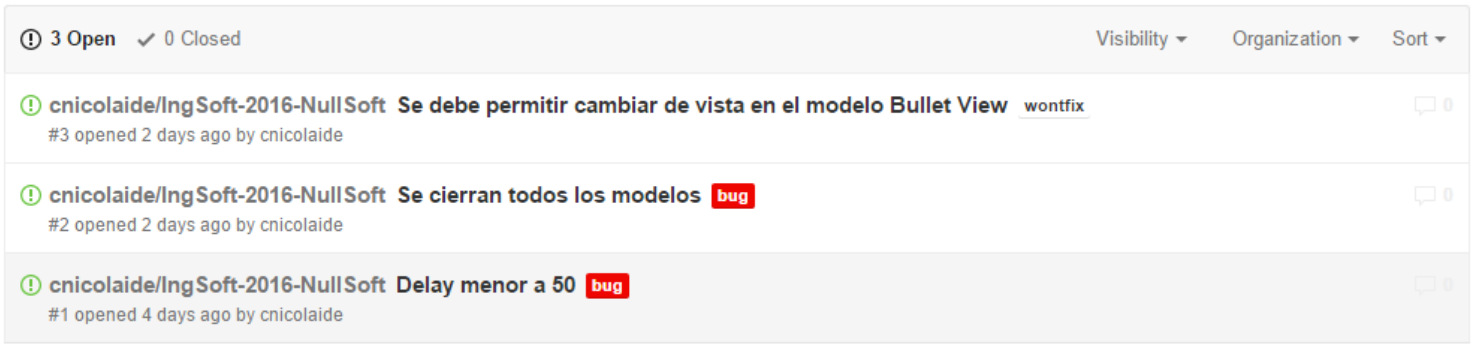
INTEGRANTES: Candotti, Enzo
Flores, Mauricio
Nicolaide, Christian

Profesor: **Mgr. Martín Miceli**
Ing. Julian Nonino

Listado de la funcionalidad incluida

Pass/Fail Ratio de sistema.

Bugs conocidos (i.e. no resueltos) en la entrega.



Lugar/link del entregable y de las instrucciones de instalación.

1. Clonar el repositorio de GitHub
2. Abrir la consola de sistema cmd
3. Hacer change directory (cd) a la carpeta del proyecto. Ej: cd C:\Users\myuser\Desktop\IngSoft-2016-NullSoft
4. Correr el comando gradlew build

Plan de Manejo de las Configuraciones

INTEGRANTES: Candotti, Enzo
Flores, Mauricio
Nicolaide, Christian

Profesor: **Mgr. Martín Miceli**
Ing. Julian Nonino

Historial de Revisiones

Versión	Fecha	Observaciones	Autor
1.0	28/05/2016		
1.1	30/05/2016		

Índice de Contenidos

1. Introducción

1.1 Propósito y alcance

SCM es el proceso mediante el cual se identifican los métodos y herramientas para controlar el software a lo largo de su desarrollo y utilización. En esta sección se describe la forma de trabajo, los documentos, el hardware, software y las herramientas utilizadas en este proyecto.

1.2 Propósito del Manejo de la Configuración del Software (SCM)

- Asegurar la consistencia de la información poniendo en práctica la SCM.
- Definir las personas que le dan soporte a las prácticas de SCM.
- Mantener la integridad a los largo de todo el ciclo de vida del producto.
- Informar a grupos y las personas el estado del proyecto.
- Crear un historial del estado anterior y actual del proyecto.
- Mejora de Procesos.

1.3 Referencias, Abreviaturas y Glosario

- **SCM** Gestión de la Configuración del Software (Software Configuration Management)
- **SVN** Sistema de control de versiones
- **CCC** Comité de Control de Cambios
- **SCI** Software Configuration item
- **SCMer** Rol encargado de realizar la tarea de gestión de la configuración (SCM).

1.3.1Conceptos básicos

Repositorio: Espacio físico (directorio y grupo de directorios), donde se almacenan, una vez terminados, todos los elementos generados durante el proceso de desarrollo en sus diferentes versiones. Estos elementos pueden estar en tres estados: Pendientes de aprobar por el área de SQA, Aprobados por SQA, y en Producción.

SCI: Software Configuration Item, Elemento de Configuración del Software. Son los elementos creados durante el proceso. Pueden ser de tres tipos: De Software: código fuente, recursos gráficos, bibliotecas, ejecutables Documentos: técnicos, administrativos y de usuario Estructuras de datos: estructura de base de datos, datos iniciales, archivos de configuración, etc.

Check Out: Tomar un SCI del repositorio y copiarlo en un área de trabajo, dejando bloqueado el SCI en el repositorio, para que nadie más pueda hacer un check out del mismo.

Check In: Si se trata de un SCI nuevo, ingresarlo en el repositorio y dejarlo disponible para posteriores modificaciones, mediante el mecanismo de check out y check in. Si se trata de un SCI existente al que se le hizo check out, el check in implica el ingreso al repositorio de una nueva versión del SCI, dejando al SCI disponible, liberando el bloqueo que registrado en el momento del check out.

2. Roles de la Administración de la Configuración

2.1 Administración de la Configuración del Proyecto

La Administración de la configuración del proyecto está a cargo del "Global PCM", él es el responsable de actividades como seguimiento de las herramientas, creación de los nuevos branches, creación del reléase, etc.

GPCM Posee toda responsabilidad sobre todos los CI. Responsabilidad en la creación de branches y administración de sus políticas. Responsabilidad y asistencia sobre el etiquetado y lanzamiento de branches. Coordinar actividades del CM en el proyecto. Asegurar la correcta ejecución del esquema del CM. Participación en auditorias. Analizar todas las novedades relacionadas al CM.

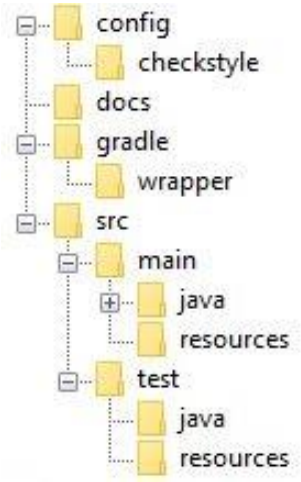
TPCM Asistencia en la creación de tags y branches. Creación de actividades para el equipo específico en branches. Garantizar la integridad del producto y el seguimiento de los elementos de configuración propios del equipo. Participación en auditorias. Analizar todas las novedades relacionadas al CM.

Team Ayudar a resolver conflictos durante la actividad de merge. Asegurarse que los criterios de calidad de los entregables a la rama principal se cumplan. Seguir todos los procesos asociados, políticas y prácticas definidas por sus roles asignados.

2.2 Dirección y forma de accesos a la herramienta de control de versiones

- **Lenguaje de Programación:** Java https://www.java.com
- **Entorno de Desarrollo:** Netbeans IDE https://netbeans.org/
- **Software de Manejo de Versiones:** Git https://git-scm.com/
- **Sistema de Control de Versiones:** GitHub https://github.com/cnicolaide/IngSoft-2016-NullSoft
- **Sistema de Seguimiento de Errores:** Git Issues https://github.com/cnicolaide/IngSoft-2016-NullSoft/issues
- **Herramienta de Integración Continua:** Travis https://travis-ci.org/cnicolaide/IngSoft-2016-NullSoft
- **Herramienta de Automatización:** Gradle http://gradle.org/

2.3 Esquema de directorios y propósito de cada uno.



Ruta	Propósito
.\config\	Contiene archivos de configuración de plugins de Gradle
.\docs\	Contiene documentación y diagramas del proyecto.
.\gradle\	Contiene el wrapper de Gradle.
.\src\main	Contiene las clases principales del proyecto.
.\src\test	Contiene las clases de test usadas en el proyecto.

2.4 Normas de etiquetado y de nombramiento de los archivos.

Para nombramiento de etiquetas se seguirá una notación numérica compuesta por tres números (y un cuarto opcional) separados por puntos con la siguiente notación:

major.minor.revision[.entrega]

Cada uno de estos números tienen el siguiente significado:

- **major**: indica la versión principal del software, consistiendo en un conjunto de funcionalidades concretas que son recogidas y cubiertas en dicha versión.
- **minor**: indican funcionalidad menor cubierta en la versión de software entregada.
- **revision**: se modifican cuando hay revisiones de código ante fallos de la aplicación.

- **entrega**: este dígito tiene el objetivo de llevar la cuenta del número de veces que una entrega se rechaza, por incumplimiento de algún requisitos de la gestión de entregas o del proyecto.

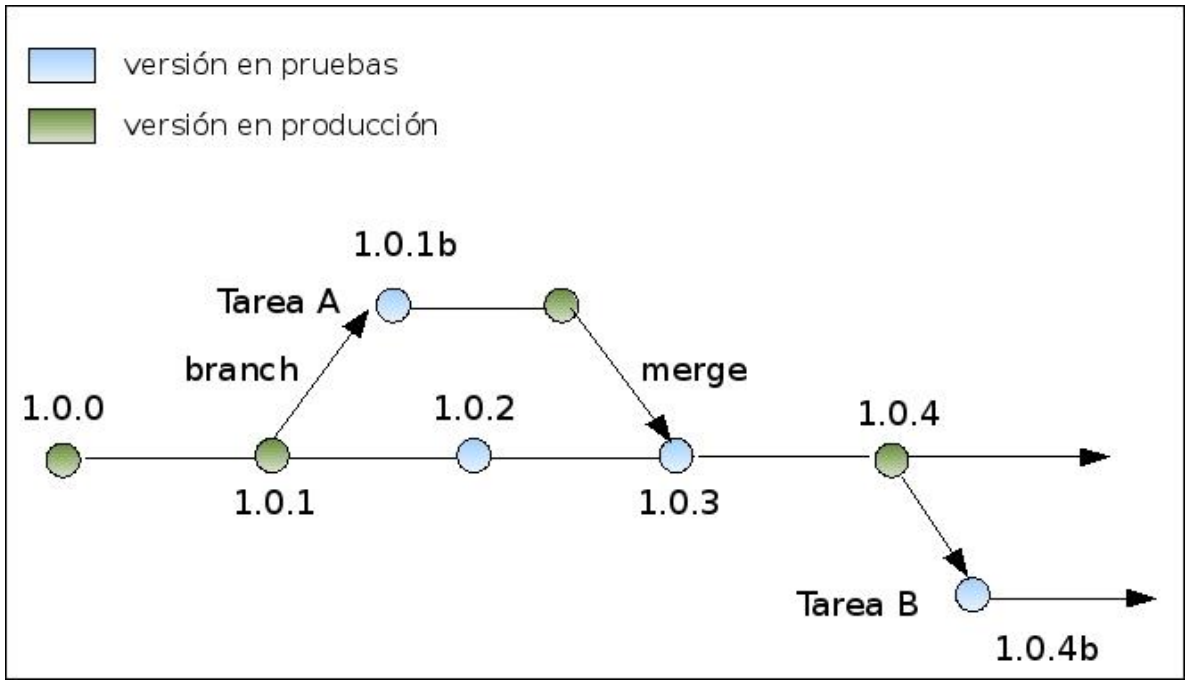
En el caso de la creación de un branch o línea de desarrollo distinta de la principal añadiremos la letra b al final de la numeración.

2.5 Plan del esquema de ramas a usar (y en uso).

Entendiendo que un branch es una línea de desarrollo distinta de la principal. Generalmente se trabajara sobre el trunk del proyecto, pero en ciertas ocasiones podrá ser necesario crear una línea de desarrollo paralela, para esto se utilizaran los branch.

Imaginando que tenemos nuestro proyecto con una linea de desarrollo principal (sobre el trunk), y que en ciertos momentos algunas de las versiones (tags) que se marcaron pasaron al entorno de producción. Pero en un determinado momento se detecta un error critico o una tarea (por ejemplo cambiar la integración con otra aplicación) y hay que abordarla de forma rápida sobre la versión que hay en producción y si pasar toda la nueva funcionalidad de las versiones que la siguieron y que no han pasado por un proceso de pruebas.

En este caso el equipo debe de crear un branch sobre el tag que marca la versión que hay en producción y sobre la que se quiere implementar alguna tarea concreta. Por ejemplo en el siguiente diagrama podemos ver que sobre la versión 1.0.1 se crea un branch 1.0.1b.



Una vez finalizada la tarea tendremos una versión que podremos desplegar (pasando por el procedimiento de entrega y pruebas previamente) en producción. Para evitar que se repitan tareas en las distintas líneas, cada branch debe representar el desarrollo una tarea concreta para

que se vuelva a integrar en la línea principal de desarrollo en poco tiempo, de esta manera la tarea de integración será menos costosa.

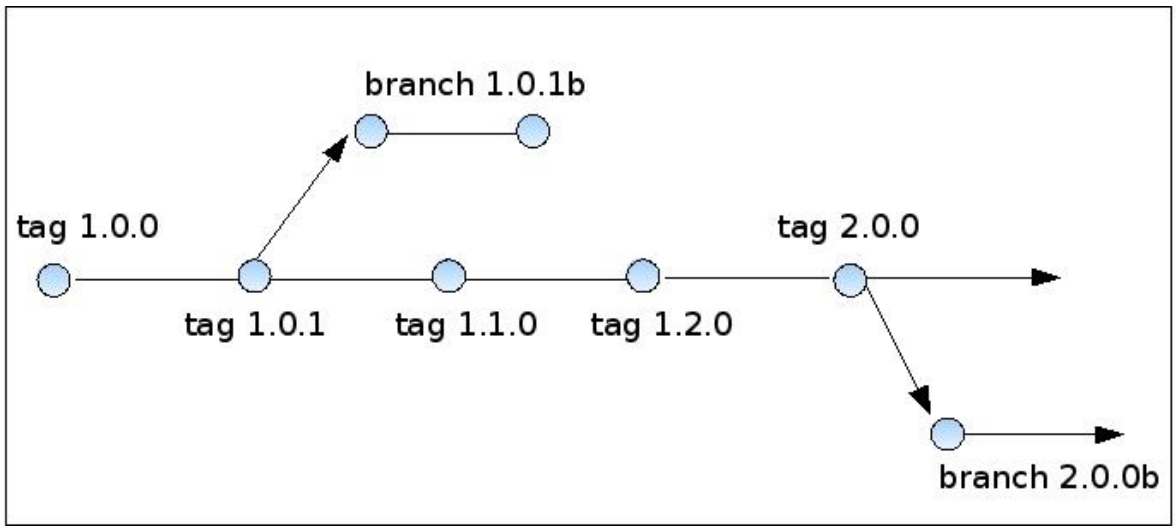
2.6 Políticas de fusión de archivos y de etiquetado de acuerdo al progreso de calidad en los entregables.

Para fusionar nuevas ramas que pudieran surgir con el código principal nuevamente, el administrador realizara la fusión haciendo un rebase del código junto a los desarrolladores. De esta forma rápidamente se podrán identificar errores, los desarrolladores podrán trabajar en resolverlos y el administrador estará al tanto de la situación para coordinar otro eventual cambio en el código.

Los criterios para modificar (incrementar) cada uno de los contadores de la etiqueta de versión son los siguientes:

- **major**: nuevas funcionalidades claves de la aplicación respecto a la versión anterior debido a la inclusión de nuevos requerimientos para el sistema, como la inclusión de nuevos módulos o una revisión completa de los existentes.
- **minor**: cambios significativos en la forma en la que se ofrece la funcionalidad existente, corrección de grandes fallos del sistema o nuevas versiones evolutivas que modifican significativamente la funcionalidad ofrecida.
- **revision**: se modifica por cada entrega de software que se realice.
- **entrega**: al rechazarse una entrega se incrementa este contador en la siguiente. Cuando la entrega se aceptase se crearía un tag público que solo conservaría los tres primeros dígitos (mayor, minor, revision).

De esta forma se puede visualizar el árbol de entregables con la siguiente estructura.



2.7 Forma de entrega de los “releases”, instrucciones mínimas de instalación y formato de entrega.

Una vez finalizadas las pruebas y llegado a un nivel aceptable de fail/ratio, se procede a la compilación del release, el compilador genera un archivo de extensión .jar, este archivo puede ser ejecutado ubicándolo en cualquier carpeta de la computadora del usuario, con la aclaración que debe estar instalado en el equipo cliente el sistema Java runtime enviroment (JRE).

El software puede ser distribuido por medio de un sitio web, CDs, o cualquier otro tipo de unidad de almacenamiento removible.

2.8 Listado y forma de contacto de los integrantes del equipo, roles en la CCB y reuniones.

El CCC es un comité que asegura que cada cambio está apropiadamente considerado por todas las partes y es autorizado antes de su implementación.

El CCC es responsable de aprobar, monitorear y controlar cada solicitud de cambios para establecer una línea de trabajo. El alcance del trabajo será aprobar o rechazar los cambios necesarios en planes, documentos y códigos. Las decisiones deberán ser tomadas respecto a las acciones que deberán estar basadas en la calidad del producto, asegurando el correcto estado del producto después de cada ciclo de prueba.

El CCC puede estar conformado por una o varias personas, pero es importante que estas tengan una visión global del proyecto. La decisión del CCC y sus apreciaciones se registran en los apartados correspondientes del formulario del cambio.

Si el CCC aprueba el cambio, se genera la asignación de la tarea de llevar a cabo dicho cambio. Esta asignación y el formulario de cambio correspondientes, llegarán al SCMer.

El formulario de cambio debe incluir:

- Cambio a realizar.
- Productos a modificar.
- Restricciones.
- Criterios de revisión y auditoría.
- Tiempo estimado para realizar el cambio y dedicación real.
- Costo estimado y real.
- Persona responsable del cambio.
- Pruebas y reportes de pruebas.

Este último punto es importante, ya que si surge un cambio se deberán realizar nuevamente las mismas pruebas de manera de verificar que no se alteró la funcionalidad existente.

Los roles de los distintos miembros estarán dados por los siguientes cargos:

- Engineering Manager
- Release Manager
- Uber Scrum Team
- GPCM

Al ser 3 integrantes en el grupo, todos realizamos diversas tareas que engloban la mayoría de las responsabilidades de los roles nombrados anteriormente.

Los datos de contacto de los miembros y los roles teóricos asignados son los siguientes:

Miembro	Email	Rol Teórico
Christian Nicolaide	cnicolaide@gmail.com	Eng. Manager, GPCM
Enzo Candotti	enzocandotti93@gmail.com	Rel. Manager
Mauricio Flores	elmauri229@gmail.com	Uber Scrum Team

Nuestra forma de trabajo consiste en 2 reuniones semanales en día y horario a convenir según disponibilidad.

Las reuniones reales las realizamos en los box de la Facultad, y para las reuniones virtuales utilizamos Skype, Facebook y WhatsApp.

2.9 Herramienta de seguimiento de bugs usado para reportar los defectos descubiertos y su estado.

Utilizamos la herramienta Issues proporcionada por GitHub para hacer seguimiento de los errores. Una vez creado el repositorio se tiene acceso a ella.

La dirección de la herramienta se encuentra definido en la sección superior.

Requerimientos

INTEGRANTES: Candotti, Enzo
Flores, Mauricio
Nicolaide, Christian

Profesor: **Mgr. Martín Miceli**
Ing. Julian Nonino

Historial de Revisiones

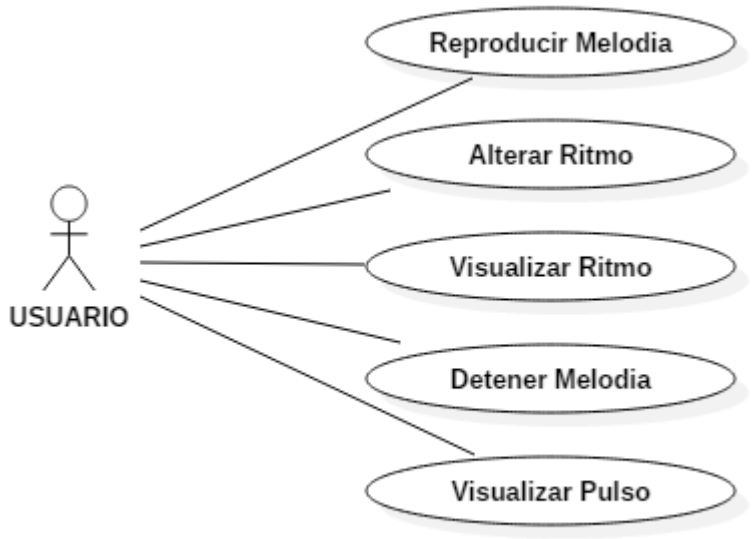
Versión	Fecha	Observaciones	Autor
1.0	30/05/2016		
1.1	31/05/2016		
1.1	10/06/2016		

Índice de Contenidos

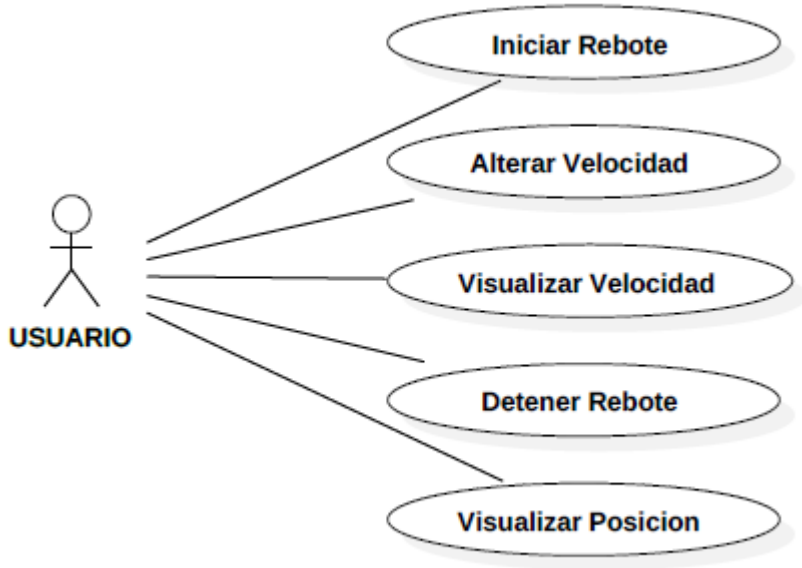
1. Diagramas

1.1 Diagramas de Casos de Usos

1.1.1 BeatModel

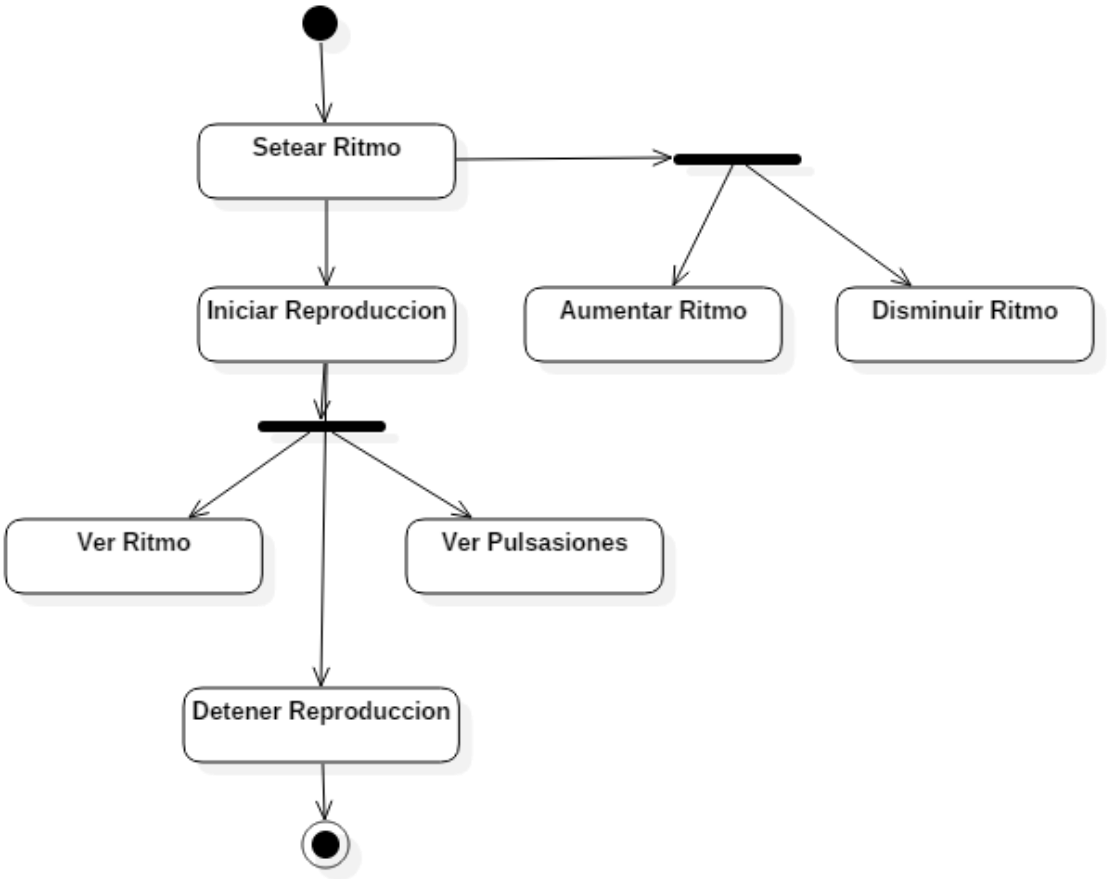


1.1.2BulletModel

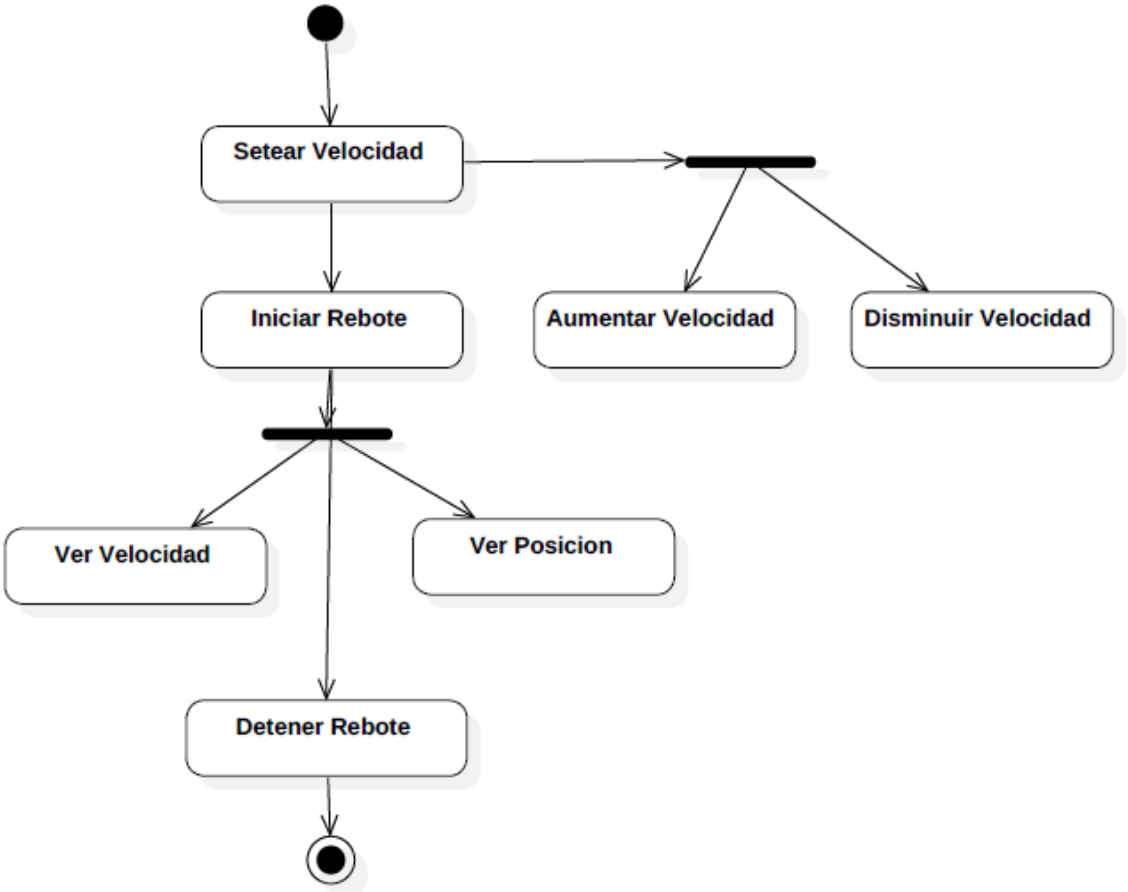


1.2 Diagramas de Actividades

1.2.1 BeatModel

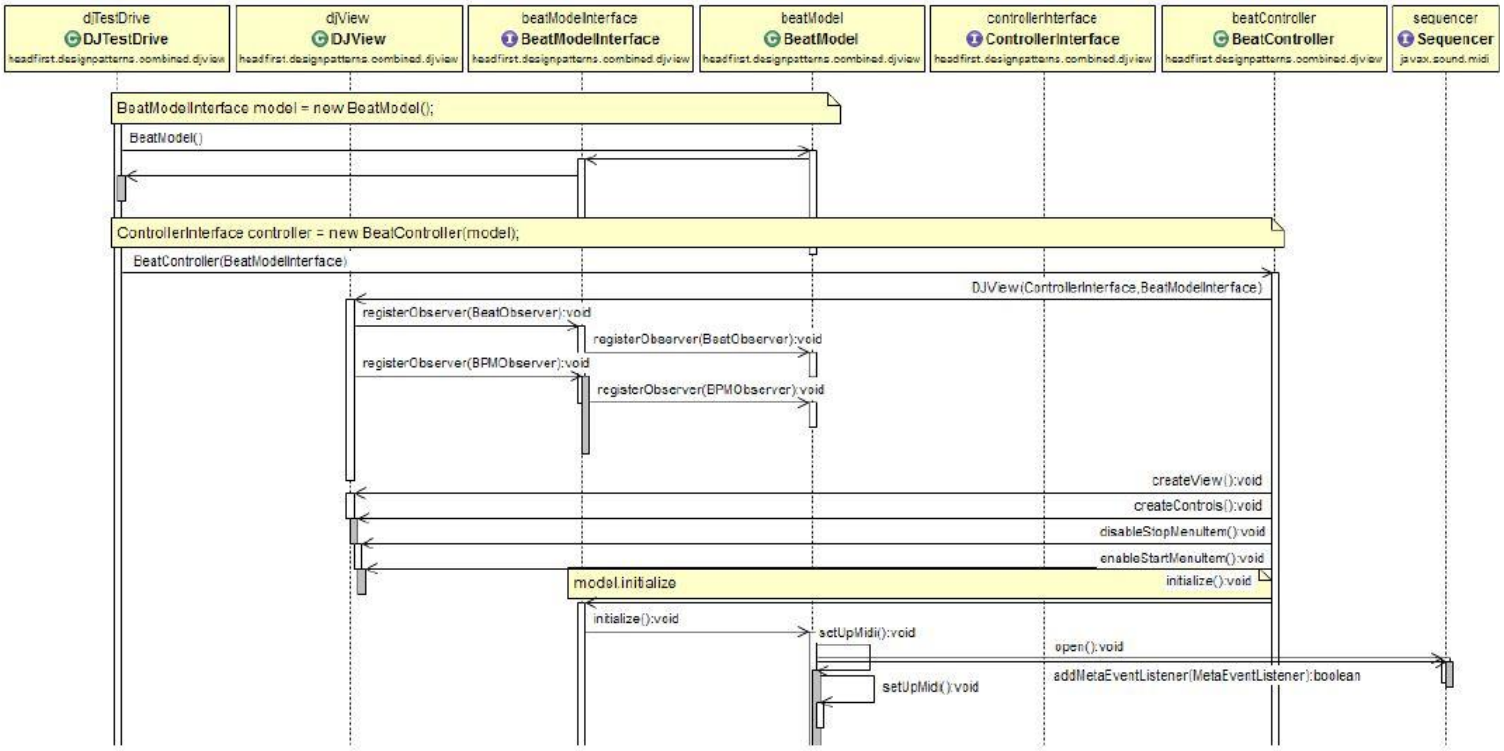


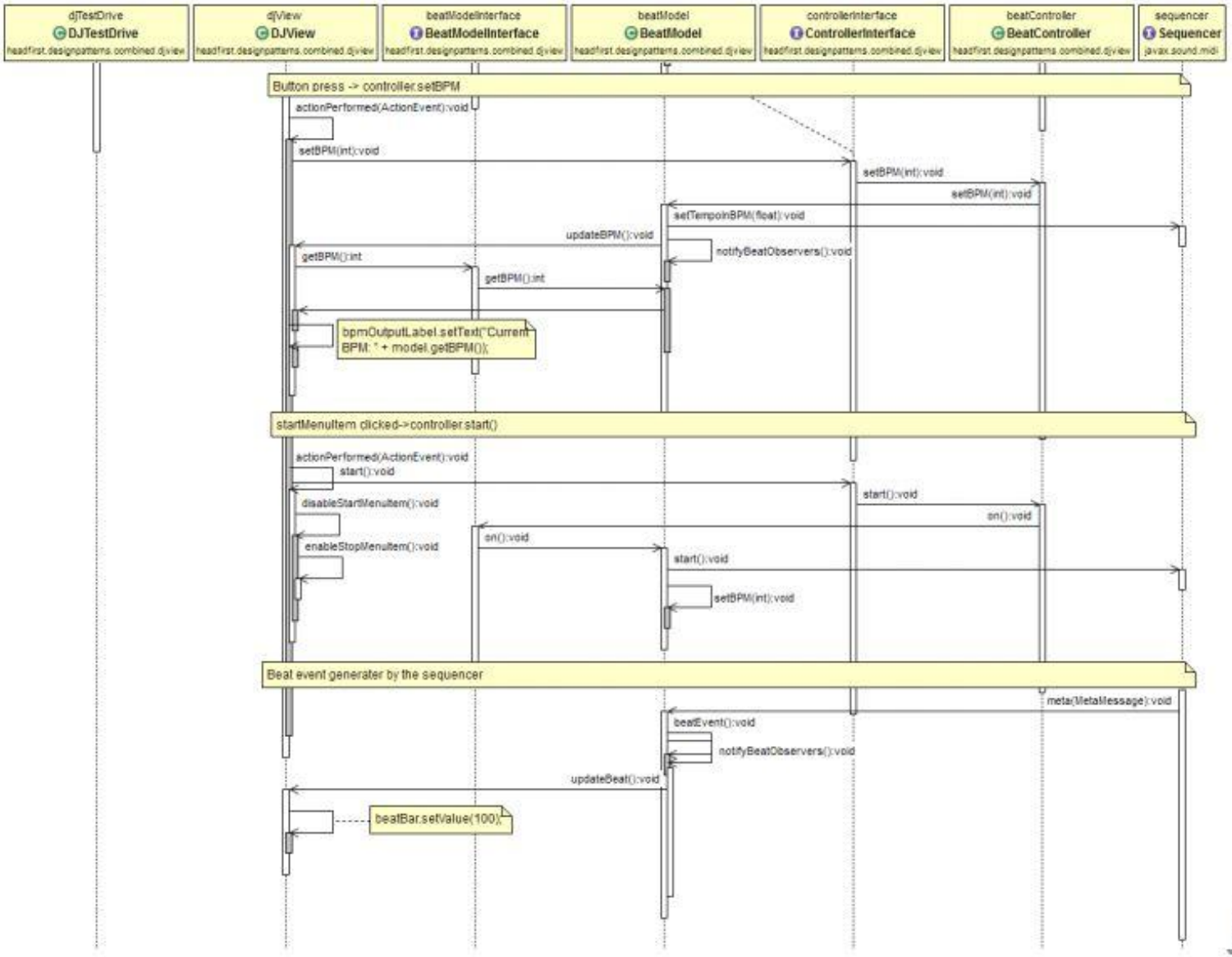
1.2.2 BulletModel



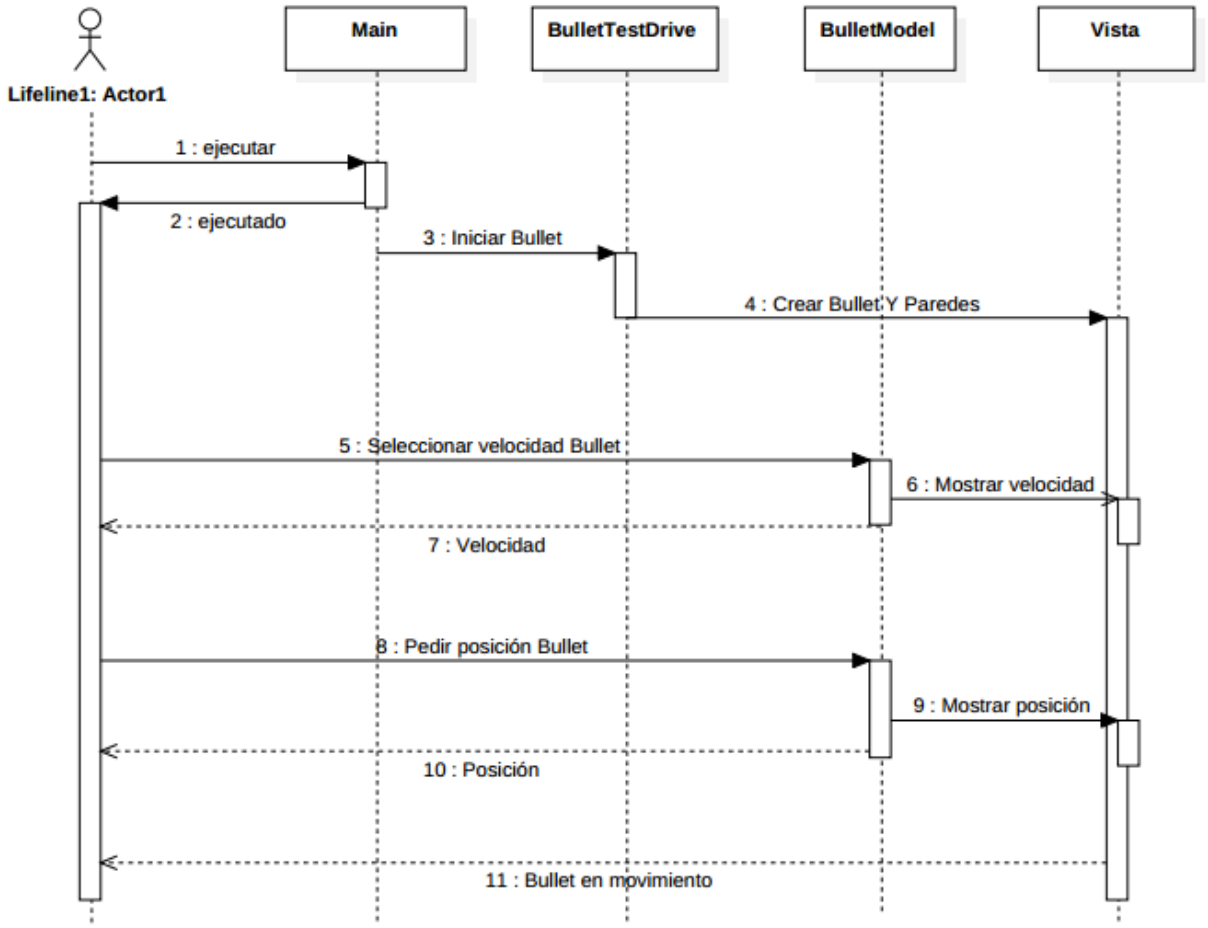
1.3 Diagramas de Secuencias

1.3.1 BeatModel


djTestDrive
DJTestDrive
headfirst.designpatterns.combined.djview
djView
DJView
headfirst.designpatterns.combined.djview
beatModelInterface
BeatModelInterface
headfirst.designpatterns.combined.djview
beatModel
BeatModel
headfirst.designpatterns.combined.djview
controllerInterface
ControllerInterface
headfirst.designpatterns.combined.djview
beatController
BeatController
headfirst.designpatterns.combined.djview
sequencer
Sequencer
javax.sound.midi
BeatModelInterface model = new BeatModel();
BeatModel()
ControllerInterface controller = new BeatController(model);
BeatController(BeatModelInterface)
DJView(ControllerInterface,BeatModelInterface)
registerObserver(BeatObserver):void
registerObserver(BeatObserver):void
registerObserver(BPMObserver):void
registerObserver(BPMObserver):void
createView():void
createControls():void
disableStopMenuItem():void
enableStartMenuItem():void
initialize():void
model.initialize
initialize():void
setUpMidi():void
open():void
addMetaEventListener(MetaEventListener):boolean
setUpMidi():void


djTestDrive
DJTestDrive
headfirst.designpatterns.combined.djview
djView
DJView
headfirst.designpatterns.combined.djview
beatModelInterface
BeatModelInterface
headfirst.designpatterns.combined.djview
beatModel
BeatModel
headfirst.designpatterns.combined.djview
controllerInterface
ControllerInterface
headfirst.designpatterns.combined.djview
beatController
BeatController
headfirst.designpatterns.combined.djview
sequencer
Sequencer
javax.sound.midi
Button press -> controller.setBPM
actionPerformed(ActionEvent):void
setBPM(int):void
setBPM(int):void
setBPM(int):void
setTempoInBPM(float):void
updateBPM():void
notifyBeatObservers():void
getBPM():int
getBPM():int
bpmOutputLabel.setText("Current BPM: " + model.getBPM());
startMenuItem clicked->controller.start()
actionPerformed(ActionEvent):void
start():void
start():void
disableStartMenuItem():void
on():void
on():void
enableStopMenuItem():void
start():void
setBPM(int):void
Beat event generater by the sequencer
meta(MetaMessage):void
beatEvent():void
notifyBeatObservers():void
updateBeat():void
beatBar.setValue(100);

1.3.2 BulletModel



2. Requerimientos Funcionales

2.1 BeatModel

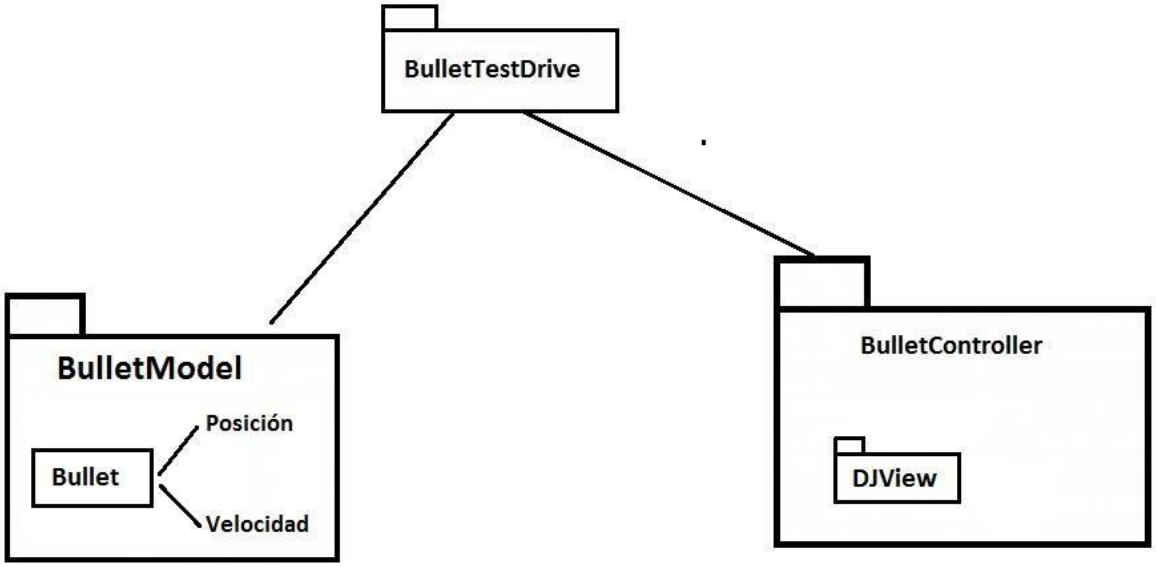
- Se reproducirá solo una pista de audio que no será posible alterar.
- El ritmo (BPM) puede ser aumentado y disminuido en valores que van de 0 a 1000.
- Se debe poder detener la reproducción de audio y reanudar tantas veces como se desee.
- El usuario debe poder observar las pulsaciones a través de una representación gráfica.
- Se debe informar el ritmo al cual se está reproduciendo la música. #### BulletModel
- Se hará rebotar una pelota contra cuatro paredes.
- La velocidad puede ser aumentada o disminuida.
- El usuario debe poder observar el movimiento de la pelota a través de una representación gráfica.
- El usuario no podra modificar la trayectoria de la pelota en el mapa.
- Se debe poder informar la velocidad y la posición de la pelota cuando esta está en movimiento.

3. Requerimientos No Funcionales

- Toda funcionalidad del sistema y transacción de negocio debe responder al usuario en menos de 2 segundos.
- El sistema debe proporcionar mensajes de error que sean informativos y orientados a usuario final.
- El tiempo de aprendizaje del sistema por un usuario deberá ser menor a 5 minutos.
- El sistema debe poseer interfaces gráficas bien formadas.
- El tiempo para iniciar o reiniciar el sistema no podrá ser mayor a 15 segundos.
- La probabilidad de falla del Sistema no podrá ser mayor a 1%.

4.0 Diagrama de Arquitectura Preliminar

4.1 BulletModel



5.0 Matriz de Trazabilidad

Casos de Uso / Requerimientos	CU1	CU2	CU3	CU4	CU5
A	x				
B		x			
C			x		
D				x	
E					x

Arquitectura, Diseño e Implementacion

INTEGRANTES: Candotti, Enzo
Flores, Mauricio
Nicolaide, Christian

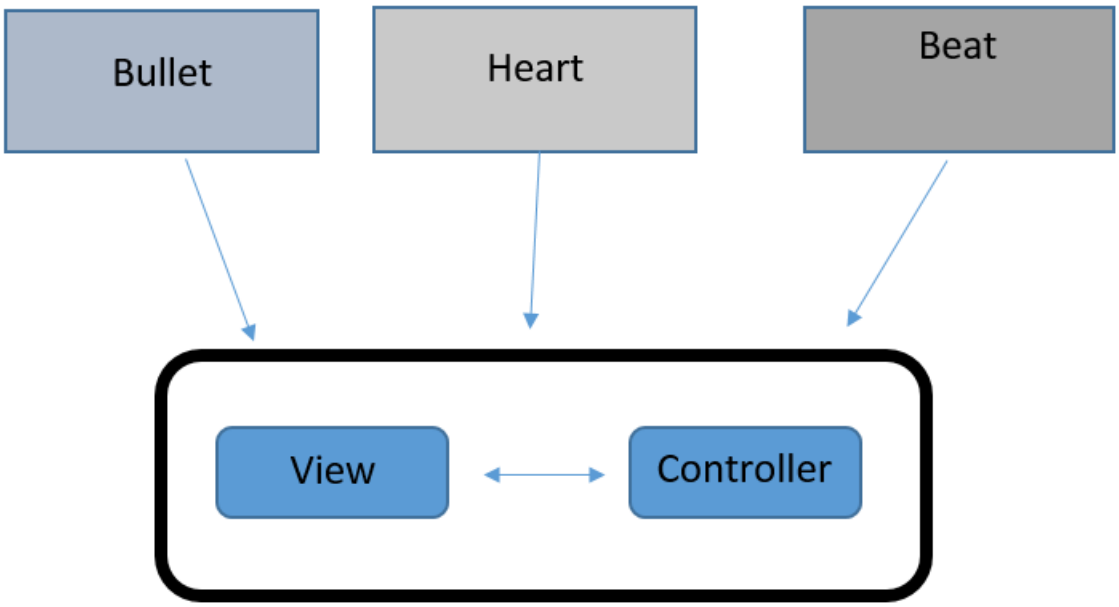
Profesor: **Mgr. Martín Miceli**
Ing. Julian Nonino

Historial de Revisiones

Versión	Fecha	Observaciones	Autor
1.0	31/05/2016		
1.1	10/06/2016		

1. Arquitectura

1.1 Patrón de Arquitectura

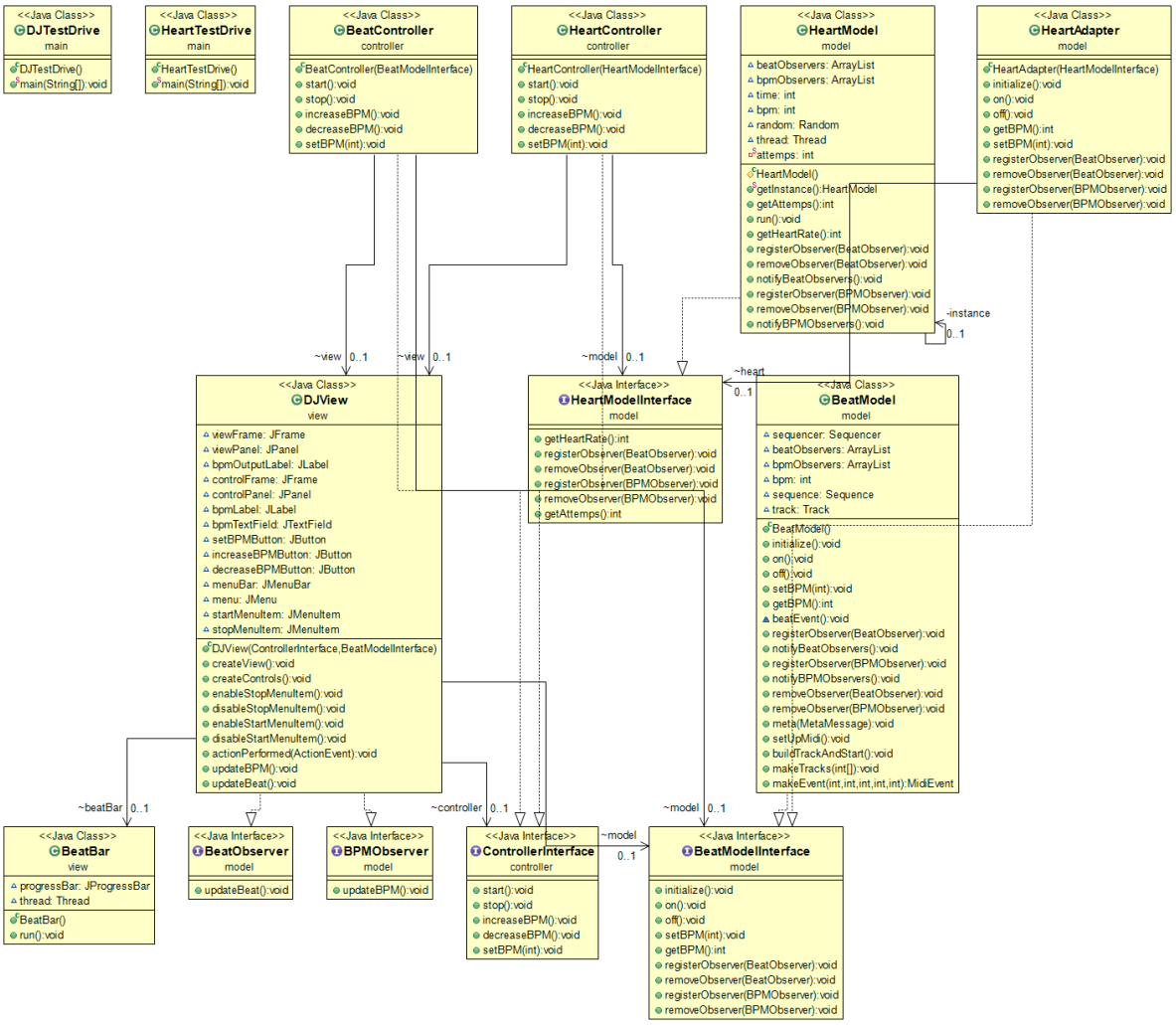


Se aplica el patrón de arquitectura MVC permitiendo independizar la lógica y la parte visual del sistema usando para eso un controlador que administra los procesos sirviendo como puente entre estos.

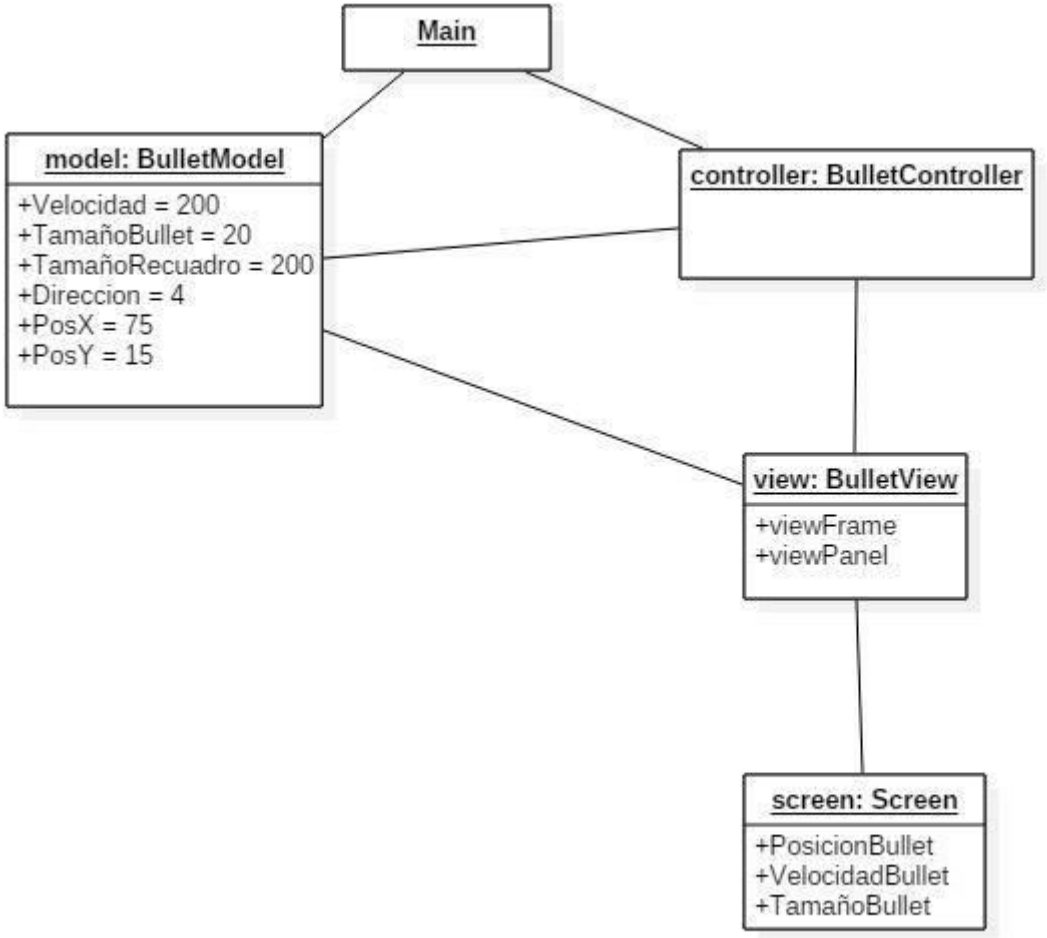
La agrupación de clases de nuestro sistema es:

2. Diseño e Implementación

2.1 Diagramas de Clase



2.2 Diagramas de Objeto



2.3 Diagramas de Paquetes